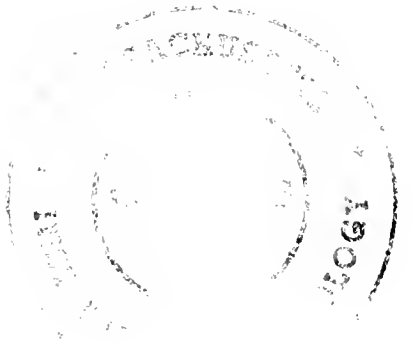

Designing Communication Networks with Hop Restrictions

Kemal Atinkemer*
and
Anantaram Balakrishnan**

Sloan W.P. No. 2080-88 July 1988

WORKING PAPER
ALFRED P. SLOAN SCHOOL OF MANAGEMENT

Designing Communication Networks with Hop Restrictions

Kemal Atinkemer*
and
Anantaram Balakrishnan**

Sloan W.P. No. 2080-88 July 1988

MASSACHUSETTS
INSTITUTE OF TECHNOLOGY.
50 MEMORIAL DRIVE
CAMBRIDGE, MASSACHUSETTS 02139

Designing Communication Networks with Hop Restrictions

Kemal Atinkemer*
and
Anantaram Balakrishnan**

Sloan W.P. No. 2080-88 July 1988

* Krannert Graduate School of Management
Purdue University
West Lafayette, Indiana

** Sloan School of Management
Massachusetts Institute of Technology
Cambridge, Massachusetts

Abstract

The topological design of a computer/communication network depends on three main considerations - investment and operating cost, transmission delays, and network reliability. Design models that explicitly account for all three criteria are complex optimization problems that can be solved only by heuristic methods. In this paper, we propose a design model that uses hop constraints to control delays and improve reliability. The hop constraints specify an upper limit on the number of channels over which messages can be transmitted. Reducing the number of hops decreases the average message delay and increases the number of alternate transmission routes; this improvement in network performance is, of course, accompanied by an increase in the total cost of the communication system. To study this tradeoff, we develop a Lagrangian relaxation-based algorithm to identify cost-minimizing network designs that satisfy all internode traffic requirements and hop restrictions. By systematically varying the hop restrictions, this method generates alternative configurations with different cost, delay, and reliability characteristics. Our model, thus, serves as a valuable decision support tool which the network designer can use to assess the cost impact of different performance specifications, and select a design that achieves the proper balance between conflicting objectives. We report results for extensive computational tests of the algorithm using several randomly generated test problems. Our results demonstrate that, even for relatively large problems, the method identifies good heuristic solutions and tight lower bounds that confirm the near-optimality of the selected designs. Using a 25-node example, we illustrate how the model can be used to evaluate the cost versus performance tradeoff.

Keywords: Computer/communication network design, cost versus performance tradeoff, Lagrangian relaxation

1. Introduction

Designing the topology of computer-communication networks is a very important and challenging problem that has received extensive attention in the communications, distributed computing and operations research literature. The importance of the topological design decision stems from the enormous capital investments required to establish communication facilities and the significant impact that this decision has on network performance measures such as delay, throughput and reliability. The design problem is very complex because of the following problem features. First, the topology of the communication network and the routing of messages on the network are closely interrelated. A densely connected network requires a significant investment in transmission facilities, but reduces message transmission delays and increases network reliability. On the other hand, a sparse network has lower cost but higher communication delays and lower reliability. The design model must account for this fundamental tradeoff by simultaneously considering the topological and routing decisions. Second, both the channel selection and routing problems involve discrete choice decisions. Consequently, the design problem formulation becomes a large-scale linear or non-linear integer programming model that is difficult to solve optimally. Third, the design decision must *satisfice* several conflicting criteria. Foremost among these criteria are (i) minimizing the total cost of investment and operating the transmission facilities, (ii) minimizing the average delay for internode traffic, and (iii) maximizing the reliability of the network, often expressed in terms of the number of alternate routes that are available between every pair of nodes. Finding the best design, therefore, involves solving a large-scale multicriteria discrete optimization model.

In this paper, we develop a useful decision support model for designing the backbone network configuration in a telecommunication system. Our model attempts to construct a minimum cost design to meet internode traffic requirements subject to some routing restrictions. Unlike previous communication network design models, we do not explicitly represent the average delay and reliability considerations. Instead, we impose a *maximum hop restriction* (or *hop constraint*) on the route between each origin-destination pair. This restriction limits the maximum number of channels (and intermediate switches) over which a message can be transmitted. From a design

point of view, hop constraints are desirable since they simplify the task of managing the traffic over the network; furthermore, they reduce the average message transmission delay and make the network more reliable. The total delay experienced by a particular message consists primarily of queueing delays at the intermediate channels and switches over which the message is transmitted; hence, limiting the number of hops controls the overall transmission delay. In addition, when we impose tight upper limits on the number of hops, the design is likely to contain several alternate paths for each origin-destination pair, thus enhancing the connectivity and reliability of the network. These improvements in performance are, of course, accompanied by corresponding increases in the network design costs. Depending on the application context, the designer must select a configuration that makes an appropriate tradeoff between cost and performance.

For a given set of hop constraints, we formulate the design problem as a mixed integer program and propose a Lagrangian relaxation method to solve it approximately (but with performance guarantees). By systematically varying the maximum hop restrictions, we can obtain a variety of design solutions that vary with respect to cost, delay, and connectivity. The network designer can then assess the cost impact of different delay and connectivity requirements before selecting an appropriate design. We discuss alternative schemes for systematically changing the maximum hop restrictions in order to generate several good solutions with different cost versus performance characteristics.

Using the hop constraints as a surrogate for delay and connectivity restrictions offers several important advantages. First, this approach greatly reduces the complexity of the model and enables us to generate provably near-optimal solutions relatively quickly using an optimization-based approach. In contrast, representing delay restrictions explicitly requires several assumptions about the distribution of traffic, and might possibly introduce non-linearities in the model. Modeling connectivity restrictions (for instance, the requirement that the network must be 3-connected) is equally complex. Consequently, models that explicitly incorporate both delay and connectivity restrictions can often be solved only by heuristic methods that do not provide any bounds on the quality of the solutions. Because we can solve the hop-constrained model effectively, our approach provides a useful design tool for performing different types of sensitivity analyses. In

particular, by changing the hop constraints and channel capacity utilization levels, the model generates several good alternate solutions which the designer can evaluate using, say, detailed simulations before selecting the most appropriate design. Our model has the advantage of not requiring any distributional assumptions or detailed delay and connectivity specifications.

The rest of this paper is organized as follows. In Section 2, we formally describe the design problem, briefly review the related literature, state our modeling assumptions, and present the mathematical programming formulation. This section also discusses alternative methods to systematically vary the maximum hop restrictions. Section 3 describes our Lagrangian-based solution approach to generate upper and lower bounds on the cost of the optimal solution, for a given set of hop constraints. In Section 4 we present extensive computational results for several randomly generated test problems. Our first set of computational tests focusses on evaluating the effectiveness (in terms of the quality of solutions generated, and computation times) of the proposed algorithm for various problem sizes, cost structures, and hop restrictions. The results demonstrate that the method constructs near-optimal solutions even for relatively large problems with up to 30 nodes and 420 edges (whose integer programming formulations contain up to 160,000 integer and continuous variables, and 800,000 constraints). Our second set of computational experiments illustrates how the model can be used to gain insights about the tradeoffs between cost, delay and connectivity by varying the hop restrictions. We analyze the solutions generated with various hop parameter settings for one sample 25-node problem. Section 5 summarizes the paper and outlines directions for further research.

Our successful experience with the hop-constrained network design approach suggests that, for many long and medium term planning contexts, using meaningful surrogate constraints in place of detailed and complex operational restrictions is a very useful modeling and analysis strategy. This approach is particularly appealing for other decision support applications such as logistics and manufacturing systems design and planning. It facilitates model solving without ignoring the interrelationship between design decisions and operational performance; its role is to provide insights into the tradeoffs between different conflicting criteria rather than generate a single solution.

2. Problem Definition and Formulation

Most large computer-communication networks have a hierarchical structure consisting broadly of a local access network and a backbone network. The *local access network* connects individual users (or subscribers, terminals, etc.) or groups of users to backbone nodes possibly through several levels of intermediate concentrators and switching centers. The *backbone network* interconnects the backbone nodes. Typically, for networks operated by communication companies, backbone nodes correspond to large metropolitan areas (sometimes, with up to 3 or 4 backbone nodes in some cities) and the backbone network consists of the long-distance communication facilities. On a smaller scale, a user organization may operate its own private computer-communication network using leased lines. In this case, the backbone nodes may correspond to geographically dispersed plants, distribution centers and administrative offices. The backbone network then interconnects the plants, warehouses and offices, while the local access network serves the communication needs within each facility. Because the transmission technology and design criteria are different for the different levels in a multilevel hierarchical network, the design task is often decomposed by level, with separate models for local access network planning (see, for example, Altinkemer and Gavish (1986, 1987)) and for backbone network design. In this paper, we focus on the backbone network design problem.

The backbone network design problem involves selecting the subset of links (or channels) to be included in the topology and determining the required capacity on each selected link in order to satisfy all internode traffic requirements and routing restrictions at minimum total cost. Observe that, in order to calculate the required channel capacities, the model must simultaneously consider both the design and traffic routing decisions. As in many previous design models, we consider a fixed (rather than adaptive), non-bifurcated routing policy (see, for example, Gerla and Kleinrock (1977)) to identify the *primary* (or most preferred) route between each origin-destination pair. (During actual operations, traffic might be routed on secondary or alternate routes if links on the primary route fail.) Several network operating systems (for example, SNA) employ this routing strategy; further, Gerla (1973) has shown that, at steady state, good fixed routing methods have channel flows and delays that are comparable to the traffic patterns and

delays for adaptive routing policies. Our model imposes routing restrictions in the form of hop constraints that specify an upper limit on the number of hops (i.e., number of channels over which messages are transmitted) in the primary route for each origin-destination pair. Transmission capacities are planned so that the channels operate at prespecified utilization levels when all traffic flows on the respective primary routes.

As we noted earlier, the backbone network design problem has been studied extensively. Rather than provide a comprehensive survey, we selectively review the recent literature to illustrate how our approach deviates from previous models and algorithms. Many authors have proposed methods to separately solve the two components of the backbone design problem, namely, the capacity assignment problem and the routing problem. The *capacity assignment* problem assumes that the traffic routes are given; for this routing pattern, the problem involves selecting the best capacity for each link from a discrete set of available channel capacities in order to minimize the total cost subject to delay restrictions. Bonucelli (1981), Chou, Ferrante, and Balagangadhar (1978), Gerla (1975), and Maruyama and Tang (1976) address this problem. On the other hand, the *routing* or *flow assignment* problem starts with a given assignment of channel capacities and seeks to identify primary routes between various origin-destination pairs in order to minimize either the average message delay or the maximum message delay. Ahuja (1979), Fratta, Gerla and Kleinrock (1973), Gavish and Hantler (1983), Gerla (1973) and Tymes (1981) develop flow assignment methods when routes are restricted to be non-bifurcated (i.e., all messages between a particular origin-destination pair must follow the same route). For bifurcated routing, Courtois and Semal (1981) discuss a heuristic which is based on Fratta, Gerla and Kleinrock's (1974) flow deviation method. Frank and Chou (1971), Cantor and Gerla (1974), Bertsekas (1980, 1984) and Yum (1981) investigate optimal route selection in networks with bifurcated routing. Gallager (1977) and Segall (1979) propose distributed algorithms for quasi-static routing, while Tcha and Maruyama (1985) focus on minimizing the maximum link utilization.

Several researchers have explored combined approaches that heuristically iterate between capacity assignment and flow assignment. Maruyama, Fratta and Tang (1977) describe a general iterative procedure that incorporates a priority traffic assignment scheme to determine a locally optimum design

satisfying the delay requirements. Gerla and Kleinrock (1977) propose a similar iterative algorithm that employs heuristic methods based on Fratta et al.'s flow deviation algorithm. Gerla, Frank, Chou and Eckl (1974) describe a cut saturation algorithm that successively increases the capacities on links belonging to saturated cutsets.

Integrated optimization-based models that simultaneously consider the capacity assignment and routing decisions have been studied by Gavish and Neuman (1986), Gavish and Altinkemer (1987) and Pirkul and Narasimhan (1987). Gavish and Neuman propose a solution method that trades off delay costs against capacity expansion costs to select routes from a prespecified set of possible routes. Gavish and Altinkemer improve this approach by implicitly considering all possible routes and, applying a better heuristic algorithm. They tested this method on problems with up to 32 nodes and 80 edges. Pirkul and Narasimhan treat average delay as a constraint rather than as an objective function component. Gavish and Neuman (1987) develop a routing and capacity assignment method that accounts for the failure probabilities of nodes and links. Recently, Pirkul and Narasimhan (1988) considered the problem of choosing primary as well as secondary routes in backbone networks.

While most of the existing design models account for delays in traffic (either as a constraint or as an objective function component), we are not aware of any optimization-based approaches to find least cost designs that simultaneously control delays and ensure adequate connectivity. Eswaran and Tarjan (1976) show that the problem of finding the least cost enhancement of existing networks to satisfy even some special types of connectivity restrictions is NP-hard.

Next, we introduce some notation, formalize the modeling assumptions, and present a mixed-integer programming formulation for the backbone network design problem with hop constraints.

2.1 Notation and Modeling Assumptions

Our backbone network design model assumes that the locations of the backbone and intermediate switching nodes are prespecified, as are the peak internode traffic requirements (say, in bits per second) that the network must accomodate. The design problem is defined over an undirected network G:(N,E)

whose vertices represent the backbone nodes and intermediate switches; the edges of this network correspond to the possible interconnections. For every pair of nodes p,q in the network, let d_{pq} represent the projected peak internode traffic (or *demand*) from p to q. We treat the traffic between each pair of nodes as a separate *commodity*, with <p,q> denoting the commodity that flows from node p to node q. Let PQ denote the set of all commodities, i.e., PQ = {<p,q> : p,q ε N with $d_{pq} > 0$}. We assume that, to serve all the traffic requirements, the selected design must be connected; otherwise, the original problem can be decomposed into smaller independent design problems. For each commodity <p,q>, the designer specifies an upper limit, denoted as h_{pq}, on the number of hops for messages originating at p and destined for q. We refer to h_{pq} as the (maximum) hop parameter for commodity <p,q>. Section 2.4 discusses alternative methods to select these hop parameters.

Insert FIGURE 1 about here

To establish transmission capacity on edge of the network, we incur a fixed cost and a variable cost that depends on the required channel capacity (see Figure 1). The *fixed cost* of edge (i,j), denoted as F_{ij}, might consist of investments for acquiring land and building the infrastructure, and other setup costs to install a communication channel between nodes i and j. For leased lines, F_{ij} represents the fixed charge paid to the communication company for using direct lines between i and j. Let v_{ij} denote the *variable* cost on edge (i,j), i.e., v_{ij} is the cost per unit of capacity (expressed in, say, bits per second) on the channel connecting i and j. This variable cost component approximates expenses for cables and other transmission equipment that can be purchased in different sizes (capacities). If the number of available sizes is limited, the true cost for transmission equipment has the shape of a step function; for design purposes, we make the simplifying assumption that the true cost function can be adequately approximated by a fixed plus linear cost curve. When the lines are leased, the variable cost corresponds to the cost per message unit charged by the communication company. As we discuss later, our model can also accomodate economies of scale and

volume discounts that are represented by piecewise linear concave cost functions.

We assume that, for each edge (i,j), the designer has prespecified a *desired capacity utilization level* denoted as ρ_{ij} (with $0 < \rho_{ij} \leq 1$) based on considerations such as length of the channel, estimates of its anticipated throughput, its criticality and so on. Thus, if the actual flow on edge (i,j) is x_{ij} units, the required capacity on this edge is $B_{ij} = x_{ij}/\rho_{ij}$. For convenience, instead of calculating the required capacity based on the actual flow, we scale the variable costs to reflect the desired capacity utilization; thus, $c_{ij} = v_{ij}/\rho_{ij}$ represents the *effective* variable cost per unit of traffic on edge (i,j) (see Figure 1). (For brevity, we henceforth refer to c_{ij} as the variable cost.) Increasing the desired capacity utilization level decreases the variable cost and vice versa. By parametrically changing the variable costs relative to the fixed costs, we can assess the cost and performance impact of alternative capacity utilization levels.

The edges in the original network G are undirected. However, to formulate the backbone network design problem, we must identify the direction of flow on each edge. For this purpose, we consider two directed arcs, denoted as {i,j} and {j,i}, corresponding to each original undirected edge (i,j). The original set of undirected edges is denoted as E; let A represent the corresponding set of directed arcs.

2.2 Mathematical Programming Formulation

Using the notation presented in the previous section, we can formulate the backbone network design problem as a mixed-integer program. Our formulation uses two sets of binary decision variables (y_{ij} and z_{ij}^{pq}) and one set of continuous variables (x_{ij}^{pq}) that are defined as follows:

$$y_{ij} = \begin{cases} 1 & \text{if edge (i,j) is included in the design} \\ 0 & \text{otherwise ,} \end{cases}$$

$$z_{ij}^{pq} = \begin{cases} 1 & \text{if the route for commodity <p,q> contains arc \{i,j\}} \\ 0 & \text{otherwise , and} \end{cases}$$

x_{ij}^{pq} = total flow of commodity <p,q> on arc {i,j}.

The y variables model the *edge selection* decision. The fixed costs serve as objective function coefficients for these variables, so that when we include an edge (i,j) in the design (by setting y_{ij} to 1) the corresponding fixed cost F_{ij} is added to the objective function value. The z and x variables represent the routing decision for each commodity; we refer to these variables as the *route selection* and *arc flow* variables, respectively. The route selection decision variables are necessary to enforce the hop constraints, while the arc flow variables account for the variable cost component in the objective function. Observe that both these variables are defined over directed arcs.

The backbone design problem can now be represented mathematically as the following mixed-integer formulation called [HCDP] (for *hop-constrained design problem*):

[HCDP]

$$\text{Minimize} \quad \sum_{(i,j)} F_{ij} y_{ij} + \sum_{\{i,j\}} \sum_{<p,q>} c_{ij} x_{ij}^{pq} \tag{2.1}$$

subject to

$$\sum_{j \in N} z_{ij}^{pq} - \sum_{j \in N} z_{ji}^{pq} = \begin{cases} 1 & \text{if } i = p \\ -1 & \text{if } i = q \\ 0 & \text{otherwise} \end{cases}$$

$$\text{for all } <p,q> \in PQ, \tag{2.2}$$

$$\sum_{\{i,j\}} z_{ij}^{pq} \leq h_{pq} \qquad \text{for all } <p,q> \in PQ, \tag{2.3}$$

$$x_{ij}^{pq} = d_{pq} z_{ij}^{pq} \qquad \text{for all } <p,q> \in PQ, \text{ all } \{i,j\} \in A, \tag{2.4}$$

$$x_{ij}^{pq} + x_{ji}^{pq} \leq d_{pq} y_{ij} \qquad \text{for all } <p,q> \in PQ, \text{ all } (i,j) \in E, \tag{2.5}$$

$$y \in \tau \text{ , and} \tag{2.6}$$

$$y_{ij},\ z_{ij}^{pq},\ z_{ji}^{pq} = 0 \text{ or } 1 \qquad \text{for all } \langle p,q\rangle \in PQ,$$

$$x_{ij}^{pq},\ x_{ji}^{pq} \geq 0 \qquad \text{all } (i,j) \in E, \qquad (2.7)$$

where τ = the set of all connected subnetworks of the original network G.

The objective function consists of minimizing the sum of the fixed design costs and the variable capacity costs. Note that we permit the variable cost c_{ij} to be different in the two directions; in fact, our subsequent solution algorithm applies even when this cost varies by commodity. Constraints (2.2) ensure that, for each commodity <p,q>, the arcs {i,j} with $z_{ij}^{pq} = 1$ form a route from p to q. These equations correspond to our assumption of a fixed, non-bifurcated routing policy for determining the primary routes. Constraint (2.3) represents the hop restriction for commodity <p,q>; it specifies that <p,q> can be routed over at most h_{pq} arcs (and hence through at most $(h_{pq}-1)$ intermediate nodes). Equation (2.4) relates the route selection and arc flow variables: if arc {i,j} belongs to the route for commodity <p,q> (i.e., if $z_{ij}^{pq} = 1$), d_{pq} units of commodity <p,q> must flow on this arc (i.e., $x_{ij}^{pq} = d_{pq}$); otherwise, this arc must not carry any flow of commodity <p,q> (i.e., $x_{ij}^{pq} = 0$). Constraints (2.5) are forcing constraints that relate the arc flow and edge selection variables: commodity <p,q> can flow in either direction on edge (i,j) only if this edge is included in the topological design. Finally, constraint (2.6) specifies that the selected design must be a connected subnetwork of the original network G. While this constraint is redundant in the integer programming formulation [HCDP], it strengthens the Lagrangian relaxation (thus improving the lower bounds) that we consider in Section 3.

Note that we have considered a design problem that is defined over an undirected network. Alternatively, the original network may be directed, with different fixed costs for using each link in the two different directions. (For instance, if transmission is permitted in only one direction on a link, the other direction effectively carries an infinite fixed cost.) The formulation for this directed problem is very similar to formulation [HCDP]; it uses directed design variables y_{ij}, and contains two forcing constraints,

one for each direction, instead of the single forcing constraint (2.5) of [HCDP]. All other constraints remain the same.

Before presenting our solution method for the hop-constrained design problem, we discuss some variants and extensions of this model and outline two methods to systematically vary the hop restrictions.

2.3 Model Variants and Extensions

Formulation [HCDP] uses the 'disaggregated' version of the forcing constraints (2.5). Alternatively, we might consider using the aggregate version of these constraints obtained by summing the left-hand side terms over all commodities <p,q>, and adding a coefficient of |PQ| to variable y_{ij} in the right-hand side. Only |A| aggregate constraints are necessary, as opposed to the |PQ|*|A| disaggregate constraints in [HCDP]. While the aggregate and disaggregate integer programming formulations are equivalent, the corresponding linear programming relaxations are different. The relaxation of the disaggregate formulation is tighter than the aggregate version (see, for example, Rardin and Choe (1979), Magnanti and Wong (1981)); hence, our formulation will generate stronger lower bounds.

In our formulation, we have used flow and route selection variables corresponding to each arc and every commodity. The hop-constrained design problem can also be formulated using path selection variables instead of the arc flow and route selection variables (Balakrishnan (1987)). For this formulation we must first enumerate all feasible origin-to-destination paths (satisfying the hop constraints) for each commodity. The formulation would contain one binary decision variable (to model the routing decision) corresponding to each possible path for every commodity; separate hop constraints are not required since our selection of feasible paths accounts for these restrictions. Each commodity would have a constraint specifying that exactly one route must be selected for that commodity. This approach might be attractive if the maximum number of hops is restricted to at most 2 or 3 for each commodity; further, the path formulation can accomodate other types of routing constraints that cannot be modeled easily in the arc flow formulation. However, when the maximum number of hops exceeds 3, the path

formulation contains an excessive number of decision variables, and we must resort to a column generation approach for solving the problem.

Our model implicitly assumes that the network design costs are primarily edge related. We can, however, also accomodate node related costs that are directly proportional to the throughput at each node. These costs might represent, for instance, investments in the switching and transmission equipment located at each backbone and switching node. If c_i represents the cost per unit of switching capacity at node i, we can incorporate this cost in the model by adding c_i to the variable arc costs c_{ji} for every arc {j,i} that is incident to node i. Further, if we use a directed network design model, we can also account for fixed node costs by splitting each node i into two nodes, say, i' and i". The enhanced network contains a directed arc {i',i"} which carries the fixed node cost. All arcs {j,i} incident to i in the original network are now incident to the 'incoming' node i'; all arcs {i,j} incident from i, now emanate from the 'outgoing' node i".

Insert FIGURE 2 about here

We can also accomodate economies of scale in capacity planning when these economies can be approximated as piecewise linear, concave cost functions, as shown in Figure 2. Suppose the cost function for edge (i,j) contains r_{ij} linear segments; let F_{ij}^r and c_{ij}^r denote, respectively, the y intercept and the slope of the r^{th} segment (see Figure 2). To model this cost function, we replace each edge (i,j) of the original network with r_{ij} parallel edges between i and j; the r^{th} edge in the enhanced network carries a fixed charge of F_{ij}^r and a variable cost of c_{ij}^r. Because the cost function is concave, the model will automatically select the appropriate cost segment corresponding to the total flow on each edge in the optimal solution (Balakrishnan and Graves (1988)).

2.4 Setting the Maximum Hop Parameters

Recall that we use the hop constraint as a surrogate for delay and connectivity restrictions. The backbone network design model's primary role

is to generate various cost minimizing solutions as the hop restrictions are parametrically changed. These solutions give the user insights about the tradeoffs between costs, delays, and connectivity. Let us first explore the relationship between hop restrictions, average delay and connectivity.

Even though the network has adequate channel capacity to handle peak traffic requirements, transmission delays arise because of queueing phenomena caused by the random arrival of messages at each node. The total delay experienced by a message originating at node p and destined for node q is the sum of the delays on all arcs (and intermediate nodes) belonging to the route from p to q. Assume, for simplicity, that the desired capacity utilization factor ρ_{ij} (specified by the user) is the same for all edges (i,j). Then, if the traffic flows satisfy the Kleinrock assumptions (see, for example, Gerla and Kleinrock (1977)), we can show that the delay per message unit (say, seconds per bit) of commodity <p,q> is directly proportional to the number of hops on the route from p to q. The *average delay* per message unit over all messages transmitted on the network is, thus, directly proportional to the weighted average number of hops over all commodities (where the weights correspond to the demands for the respective commodities). This simplified analysis demonstrates the direct effect that hop restrictions have on the average delay: as we reduce the upper limit on the number of permissible hops the average delay should decrease and vice versa. For general traffic distributions this relationship might not be linear, however.

In addition to decreasing the average delay, reducing the maximum number of hops also increases the connectivity since it makes the design more dense. We define connectivity in terms of the number of arc-disjoint paths between various origin-destination pairs. This number serves as a lower bound on the number of transmission links that must fail before the origin and destination are disconnected. (Alternatively, to account for node failures, we might use the number of alternate node-disjoint paths as a measure of connectivity.) At one extreme, if all commodities have a maximum hop restriction of n (where n is the number of nodes in the network) and if the variable costs are negligible, then the optimal design will be the minimal spanning tree (with respect to the fixed edge costs); this design is 1-connected and is the least reliable. On the other hand, if all commodities have a maximum hop restriction of 1 (and assuming that commodities are defined for all possible

node pairs p,q), the only feasible design is a complete network; this design has the maximum possible connectivity (of (n-1) for all commodities).

Therefore, from a network performance point of view, reducing the maximum number of hops h_{pq} for one or more commodities improves the operating performance. However, by decreasing h_{pq} we make the hop constraint more restrictive and, consequently, increase the cost of the network. What we require is a set of alternative solutions that differ in their cost and performance characteristics so that the designer can assess the cost impact of different performance specifications, and select a solution that makes the 'best' cost versus performance tradeoff. To generate these alternative solutions we must systematically vary the hop restrictions. We propose two alternative schemes for this purpose.

The first scheme, which we call the *Uniform Hop Restriction Method*, uses the same hop constraint parameter for all commodities, i.e., $h_{pq} = h_0$ for all commodities <p,q>. As h_0 increases from 1 to n, the total cost should decrease (since the hop constraints become less restrictive) while the performance deteriorates.

Our second method, called the *Commodity-dependent Hop Restriction Method*, sets different hop parameters for different commodities. In particular, the method generates more stringent hop restrictions (i.e., lower h_{pq} values) for commodities with higher demands and vice versa. The method is based on the following rationale. As we showed earlier, the average delay over all messages in the network is approximately proportional to the weighted average number of hops over all selected commodity routes. For any given vector $h = \{h_{pq}\}$ of hop parameters, let h'_{pq} ($\leq h_{pq}$) denote the actual number of hops for commodity <p,q> in the optimal solution to [HCDP]. We then define the *weighted average number of hops*, denoted as $\hat{h}$, in terms of h'_{pq} as follows:

$$\hat{h} = \sum_{<p,q>} d_{pq} h'_{pq} \Big/ \sum_{<p,q>} d_{pq} . \qquad (2.8)$$

Observe that the commodity demands d_{pq} serve as the weights for calculating $\hat{h}$. Thus, commodities with large demands contribute more to the average number of hops (and hence to the average delay). We should, therefore, impose more

stringent hop restrictions on high demand commodities. Our commodity-dependent hop restriction method mechanizes this strategy. The method requires the user to specify a *window* or range of permissible maximum hop parameters. Let $\underline{h}$ and $\bar{h}$ denote, respectively, the smallest and largest desired maximum hop parameter. If commodity <p,q> has the largest demand among all commodities, the commodity-dependent method sets the value of h_{pq} equal to $\underline{h}$; if <p,q> has the smallest demand, then h_{pq} is set equal to $\bar{h}$. For commodities with intermediate demand values, the hop parameter is calculated by linear interpolation (rounded down to the next lower integer value) in the range $[h_1, h_2]$.

Observe that the second method generalizes the first; by setting $h_1 = h_2 = h_0$ we obtain the hop parameters corresponding to the uniform hop restriction method. For the commodity-dependent method, the user can vary two parameters: the location of the window and its width. With narrow windows and values of h_1 close to 1, we obtain high cost, high performance network designs. On the other hand, when h_1 is close to n, the cost decreases but the delay and reliability performance measures deteriorate. In Section 4.3, we present computational results to illustrate the effect of various hop parameter settings using these two methods. We next discuss a Lagrangian-based algorithm for generating good upper and lower bounds on the optimal value of [HCDP] for a given set of hop parameters.

3. Solving the Hop-constrained Design Problem

The hop-constrained design problem generalizes the fixed-charge network design problem which itself is known to be NP-hard (Johnson, Lenstra and Rinnooy Kan (1978)). Finding the optimal hop-constrained solution is, therefore, a difficult computational task. In this section, we develop a Lagrangian-based method that simultaneously generates good feasible solutions and lower bounds on the optimal cost, for any given set of hop parameters. While the method is essentially an optimization-based heuristic approach, the bounds that it generates provide a performance guarantee so that the user can assess the quality of the heuristic solutions.

3.1 Lagrangian Relaxation Scheme

The Lagrangian relaxation method has been successfully applied to several difficult discrete optimization problems (see, for example, Geoffrion (1974), Fisher (1981)). The method exploits special structure in the problem formulation by dualizing complicating constraints (i.e., by removing them from the constraint set and instead incorporating them in the objective function); the resulting subproblems can then be solved efficiently using specialized algorithms. For a given set of Lagrange multipliers, the cost of the optimal Lagrangian subproblem solution serves as a lower bound on the optimal cost of the original problem. The Lagrangian subproblem solutions can also be used to construct feasible solutions for the original problem. The gap between the Lagrangian lower bound and the cost of the best feasible solution measures the maximum possible cost differential between the optimal and heuristic solutions. To generate good lower bounds, the Lagrange multipliers are changed iteratively using techniques such as subgradient optimization or dual ascent (see, for example, Fisher (1981)).

For the hop-constrained design problem, we consider a Lagrangian relaxation scheme that dualizes the forcing constraints (2.5) using nonnegative Lagrangian multipliers μ_{ij}^{pq} for all arcs $\{i,j\}$ and all commodities $\langle p,q\rangle$. Observe that when constraints (2.5) are removed from formulation [HCDP], the problem decomposes into two main subproblems: a *Routing* subproblem denoted as [RSP(μ)], and an *Edge Selection* subproblem denoted as [ESP(μ)]. As the names suggest, the routing subproblem determines the optimal values of the

route selection (z) and arc flow (x) variables, while the edge selection subproblem calculates the values of the design variables (y). The routing subproblem further decomposes by commodity; we denote the subproblem corresponding to commodity <p,q> as $[RSP_{pq}(\mu)]$. The next two sections describe these subproblems in greater detail and discuss methods to solve them.

3.2 Routing Subproblem

For any given set of Lagrange multipliers $\mu = \{\mu_{ij}^{pq}\}$, let

$$\tilde{c}_{ij}^{pq} = c_{ij} + \mu_{ij}^{pq} \qquad \text{for all } \{i,j\} \in A, \text{ all } \langle p,q\rangle \in PQ. \tag{3.1}$$

We refer to $\tilde{c}_{ij}^{pq}$ as the *adjusted* variable cost for commodity <p,q> on arc {i,j}. Then, the routing subproblem $[RSP_{pq}(\mu)]$ corresponding to commodity <p,q> has the following formulation:

$[RSP_{pq}(\mu)]$

$$Z_{pq} = \text{Min} \sum_{\{i,j\}} \tilde{c}_{ij}^{pq} x_{ij}^{pq} \tag{3.2}$$

subject to

$$\sum_{j \in N} z_{ij}^{pq} - \sum_{j \in N} z_{ji}^{pq} = \begin{cases} 1 & \text{if } i = p \\ -1 & \text{if } i = q \\ 0 & \text{otherwise} \end{cases} \tag{3.3}$$

$$\sum_{\{i,j\}} z_{ij}^{pq} \leq h_{pq} \tag{3.4}$$

$$x_{ij}^{pq} = d_{pq} z_{ij}^{pq} \quad \text{for all } \{i,j\} \in A, \text{ and} \tag{3.5}$$

$$z_{ij} = 0 \text{ or } 1$$
$$x_{ij}^{pq} \geq 0 \quad \text{for all } \{i,j\} \in A. \tag{3.6}$$

Observe that subproblem $[RSP_{pq}(\mu)]$ essentially seeks that shortest directed path from node p to node q containing a maximum of h_{pq} arcs; the adjusted variable costs $\tilde{c}_{ij}^{pq}$ serve as arc lengths for this *hop-constrained shortest path problem*. Setting $z_{ij}^{pq} = 1$ and $x_{ij}^{pq} = d_{pq}$ for all arcs $\{i,j\}$ belonging to the hop-constrained shortest path (and all other z and x variables corresponding to commodity <p,q> equal to zero) gives the optimal solution to $[RSP_{pq}(\mu)]$. The optimal value $Z_{pq}(\mu)$ of this routing subproblem is equal to d_{pq} times the length of the shortest hop-constrained path.

We solve the hop-constrained shortest path subproblem using a truncated version of the method of successive approximations (pp. 74, Lawler (1976)). (Saigal (1968) and Rosseel (1968) discuss a similar method for finding the shortest path containing <u>exactly</u> h arcs.) This algorithm solves the problem correctly when the network does not contain any negative cycles. If we assume that all variable costs c_{ij} are nonnegative, then the adjusted variable costs are also nonnegative (since the Lagrange multipliers μ_{ij}^{pq} are nonnegative, by definition); hence, our network does not contain any negative cycles. The

method proceeds as follows: let u_j^h denote the length of the shortest path, containing *at most* h arcs, from node p (the origin) to node j. Initially, $u_j^1 = \tilde{c}_{pj}^{pq}$ if $(p,j) \in E$; otherwise $u_j^1 = \infty$. At the end of stage h, the algorithm has identified u_j^h for all nodes j in the network. We use the following expression to compute u_j^{h+1} for all $j \in N$ at stage (h+1):

$$u_j^{h+1} = \text{Min} \{ u_j^h, \text{Min} [u_j^{h-1} + \tilde{c}_{ij}^{pq} : i \in N, \{i,j\} \in A] \}. \qquad (3.7)$$

The first term in the right-hand side of equation (3.7) represents the length of the p-to-j shortest path with at most h arcs. The second term expresses the fact that the (h+1)-arc shortest path from p to j must consist of a h-arc shortest path from p to i, for some intermediate node i, plus the arc from i to j. Observe that if we run the algorithm for (n-1) stages (where n is the number of nodes in the network), the algorithm determines the lengths of the unconstrained shortest paths from node p to all other nodes of the network.

To solve subproblem $[RSP_{pq}(\mu)]$, we terminate the algorithm after h_{pq} stages; the method's computational complexity is $O(n^2 h_{pq})$. The value $u_q^{h_{pq}}$ that is obtained at the end of the last stage (i.e., with $h = h_{pq}$) multiplied by the demand d_{pq} gives the optimal value of the routing subproblem. We can trace the actual hop-constrained shortest path for commodity <p,q> by performing the usual backtracking procedure.

3.3 Edge Selection Subproblem

Given a set μ of Lagrangian multipliers, let

$$\tilde{F}_{ij} = F_{ij} - d_{pq} \mu_{ij}^{pq} \qquad \text{for all edges } (i,j) \in E, \qquad (3.8)$$

represent the *adjusted* fixed cost for edge (i,j). The edge selection subproblem involves selecting a subset of edges that connects all the nodes of the network at minimum total (adjusted fixed) cost. We solve this subproblem as follows: First, select all edges whose adjusted fixed costs are negative, and arrange the remaining edges in order of nondecreasing adjusted fixed cost. At each stage we refer to the subnetwork defined by the currently selected edges as the *current* network. Consider each arc (i,j) in sequence from the list of unselected edges. If edge (i,j) connects two different components of

the current network, then select it and update the current network by merging the two components; otherwise, discard edge (i,j) and consider the next edge in the list. The procedure terminates when the current network consists of a single connected component. The sum of the adjusted fixed costs for the selected edges gives the optimal value of the edge selection subproblem. The computational effort required to solve this subproblem is determined by the edge sorting procedure which requires O(mlogm) operations, where m denotes the number of edges in the original network.

The optimal value $\underline{Z}(\mu)$ of the complete Lagrangian subproblem is the sum of the edge selection subproblem value and the optimal values of the route selection subproblems for all commodities. For any given set of nonnegative multipliers μ, $\underline{Z}(\mu)$ is a lower bound on the optimal value of the original subproblem. To obtain good lower bounds, we use the subgradient method to iteratively adjust the Lagrange multipliers. Briefly, the method increases multiplier μ_{ij}^{pq} if the current Lagrangian subproblem solution violates the forcing constraint (2.5) corresponding to edge (i,j) and commodity <p,q>; if this constraint is satisfied as a strict inequality, the multiplier is reduced to a lower nonnegative value; otherwise, the multiplier is unchanged. The user must specify an initial step size multiplier value that is used for computing the amount by which the multipliers are increased or decreased at each iteration. The method does not guarantee monotonic increase in the lower bound from one iteration to the next; if the lower bound does not increase for a prespecified number of consecutive iterations, the current step size multiplier value is halved. The procedure is terminated either after executing a user-specified maximum number of iterations or if the gap between the upper and lower bounds is below a given tolerance level.

We note that our Lagrangian relaxation scheme does not satisfy the *integrality* property (Geoffrion (1974)), i.e., if the subproblems are solved as linear programs, the optimal solutions might possibly be fractional. Therefore, the best lower bound that can be obtained using our relaxation scheme is likely to be better than the linear programming lower bound for formulation [HCDP]. Of course, the subgradient method is only a heuristic procedure for solving the Lagrangian dual problem; hence, it may not generate the best possible lower bound.

Finally, we might also consider alternative Lagrangian relaxation schemes that dualize other constraints of formulation [HCDP] instead of the forcing constraints (2.5) that our method dualizes. For instance, consider a scheme that dualizes the hop constraints (2.3). The resulting Lagrangian subproblem is a fixed-charge network design problem, which is difficult to solve optimally (since it is NP-hard). However, we can solve it approximately using, say, a dual ascent procedure (Balakrishnan, Magnanti and Wong (1988)). The dual ascent method generates a lower bound on the cost of the optimal subproblem solution; this value also serves as a lower bound for the original (hop-constrained) problem. Unfortunately, the Lagrangian subproblem solution for this scheme does not readily provide a feasible design satisfying the hop constraints. On the other hand, our method generates a feasible starting solution that can be improved using a local improvement heuristic; we discuss this heuristic procedure next.

3.5 Lagrangian-based Heuristic Procedure

At each subgradient iteration, the solution to the routing subproblem provides a set of feasible routes (that satisfy the hop restrictions) for all commodities. Note that the edge selection subproblem may not select all the edges belonging to these routes; alternatively, the subproblem may include more edges than are necessary for the selected routes. We, therefore, ignore the solution to the edge selection subproblem, and use only the routing subproblem solution to construct an initial design that is feasible.

Our Lagrangian-based initial design consists of all edges that belong to the routes selected in the routing subproblem. For this design, we solve the hop-constrained shortest path problem for each commodity, using the <u>original</u> variable costs c_{ij} as arc lengths. The sum of the fixed costs for the selected arcs and the variable costs for all commodities gives the cost of the heuristic solution.

We then attempt to improve this starting solution by applying a *Drop* heuristic (Billheimer and Gray (1973)). The drop procedure is a local improvement method that iteratively eliminates existing edges from the current design so that the upper bound decreases monotonically. When we drop an edge, the total fixed cost decreases; however, the variable costs might increase

since some commodities must now flow on more expensive alternate routes. At every stage, the procedure evaluates the net savings Δ_{ij} obtained by dropping each edge (i,j) from the current design. If the net savings is nonpositive for all edges, the procedure stops. Otherwise, it drops an edge with positive net savings, updates the current design and commodity routings, and decreases the upper bound. When several edges give positive net savings, we can employ one of many possible methods to select the edge that must be dropped. For instance, a 'greedy' approach would drop the edge that results in the maximum savings at each iteration. Alternatively, we might examine edges in some prespecified order and select the first edge that gives net positive savings. When the procedure terminates, the current best upper bound and the current incumbent are updated if the new heuristic solution has a lower total cost.

Evaluating the total variable cost when an edge is dropped entails solving a hop-constrained shortest path problem for each commodity. (If the variable costs are the same for all commodities, some computational savings can be obtained by simultaneously finding the hop-constrained shortest paths for all commodities that originate at a common node through a single application of the hop-constrained shortest path algorithm.) Since this computation can be very time-consuming, we do not evaluate the savings for every edge at each iteration. Instead, we maintain and iteratively update a *candidate list* of edges to evaluate. Initially, this list consists of all edges that yield net savings in the first iteration. In subsequent iterations, we only evaluate the savings for edges that belong to this list; any edge that does not give net savings in the current iteration is removed from the list. The edge with the maximum savings is dropped from the current design and also removed from the list. When the list becomes empty, we reinitialize it by evaluating the savings for all edges in the current design.

The local improvement heuristic can be applied at the end of each subgradient iteration. However, to reduce computation time, we apply the drop procedure only when the starting solution derived from the current routing subproblem has a lower cost than the previous best starting solution.

The next section describes our computer implementation of the Lagrangian-based solution procedure, and presents extensive computational results for several randomly generated test problems.

4. Computational Results

We implemented the Lagrangian-based algorithm for the hop-constrained design problem in standard FORTRAN on an IBM 3083 (model BX) computer. We tested the algorithm using several randomly generated problems. This section first describes some features of our implementation, and the method we used to generate random test problems. In all, we applied the algorithm to over 65 problem instances. Our computational tests (discussed in Sections 4.3 and 4.4) address two different performance aspects. The first set of tests focusses on evaluating the performance of the algorithm, in terms of computation time and quality of the Lagrangian-based heuristic solutions, as a function of the problem size, cost structure, and hop restrictions. We then assess the impact on cost and network performance when the hop restrictions are systematically varied (using the two methods described in Section 2.4) for one particular problem instance. This exercise demonstrates the effectiveness of our approach for generating alternative solutions with different cost, delay and connectivity characteristics.

4.1 Implementation Details

In addition to the features described in Section 3, our implementation of the Lagrangian relaxation algorithm incorporates (i) a method to generate an initial upper bound before initiating the subgradient procedure, (ii) three alternative multiplier initialization methods, (iii) several user-specified options for the subgradient procedure, and (iv) a modified rule for applying the drop procedure at intermediate stages of the subgradient procedure.

4.1.1 Generating the Initial Upper Bound

To generate an initial upper bound, we first construct a feasible design using a method which we call the *BUILD* heuristic, and subsequently apply the drop procedure to improve this design. As the name suggests, the Build heuristic progressively develops a feasible solution by sequentially considering the different commodities. The method operates as follows. The commodities are first sorted in decreasing order of demand. The main design construction procedure consists of $|PQ|$ stages, one corresponding to each commodity. At each stage, the method augments the current design by adding

edges to ensure that the next commodity has a feasible path (satisfying the corresponding hop constraint). Let E^{k-1} denote the set of edges in the current design at the beginning of stage k. (Initially (i.e., for k = 1), the set E^0 is empty.) Suppose commodity <p,q> is the k^{th} commodity in the sorted list. The algorithm seeks to find or create a good feasible route for this commodity during stage k. For this purpose, we solve a hop-constrained shortest path problem from node p to node q (containing a maximum of h_{pq} arcs) with arc lengths l_{ij} defined as follows:

$$l_{ij} = \begin{cases} c_{ij} & \text{if } (i,j) \in E^{k-1} \\ c_{ij} + (F_{ij}/d_{pq}) & \text{if } (i,j) \notin E^{k-1}, \end{cases} \qquad (4.1)$$

for all edges (i,j) of the original network. Essentially, we assign to commodity <p,q> a variable cost of c_{ij} if edge (i,j) already belongs to the current design. Otherwise, we force this commodity to absorb the entire fixed cost of edge (i,j) by adding a per unit cost of F_{ij}/d_{pq}. Let P^k denote the set of edges belonging to the shortest hop-constrained path from p to q using arc lengths l_{ij}. The current design is then augmented by adding to E^{k-1} all the new edges of P^k, i.e., we set $E^k = E^{k-1} \cup P^k$. Thus, at the end of stage k, the current design contains feasible paths for the first k commodities in the sorted list. Note that, as the algorithm progresses, fewer edges should be added to the current design at each stage since the likelihood of this design containing feasible routes for the remaining commodities increases. Since high-demand commodities contribute more to the variable cost, we give them greater importance in developing the design by considering commodities in order of decreasing demand. When the procedure terminates, the set of edges $E^{|PQ|}$ constitutes a feasible design. As a final step, the algorithm reroutes all commodities over the respective shortest hop-constrained paths using the original variable costs c_{ij} as arc lengths for edges (i,j) belonging to the feasible design.

We then apply the drop procedure (described in Section 3.5) to the starting solution constructed by the Build method. The improved solution generated by the drop heuristic provides the initial upper bound and initial incumbent (before applying the subgradient procedure).

Instead of applying the Build method to construct a starting design, we also considered using the 'all-inclusive' design, consisting of all edges in the original network, to initiate the drop procedure. We found that, in this case, many more edges had to be dropped before reaching a local optimum; hence, the computation time increased by several orders of magnitude. Since we are only interested in generating an approximate initial upper bound for purposes of calculating the subgradient step sizes, we decided to use the Build method for all our computational tests.

4.1.2 Multiplier Initialization Methods

We considered three alternative methods to initialize the Lagrange multipliers μ_{ij}^{pq}. The first method initializes all multipliers to zero. The second method sets $\mu_{ij}^{pq} = F_{ij}/D$ for all $(i,j) \in E$ and all $\langle p,q \rangle \in PQ$, where D denotes the total demand for all commodities. The justification for this method is as follows. Intuitively, if commodity $\langle p,q \rangle$ uses edge (i,j), the multiplier μ_{ij}^{pq} is a 'variable' cost component that represents the share of the fixed cost F_{ij} that each unit of $\langle p,q \rangle$ should absorb. Since at most D units can flow on edge (i,j), the second method initially allocates a cost of 1/D times the fixed cost F_{ij} to each unit of every commodity (effectively assuming that all commodities will flow on all edges). Our third initialization method uses the initial heuristic solution to determine the starting values for the multipliers. In this method, we first calculate the flow, say, x_{ij} on each edge (i,j) of the initial heuristic solution. The multiplier μ_{ij}^{pq} is then set equal to F_{ij}/x_{ij} if commodity $\langle p,q \rangle$ uses edge (i,j) in the initial solution; otherwise, this multiplier is initialized to zero.

After some initial experimentation, we found that none of the three methods dominated the others in terms of the total computation time and quality of the final upper and lower bounds; method 1 seemed to perform worse than the other two methods in many instances. We, therefore, decided to use the third (initial heuristic-based) multiplier initialization method for all our computational tests.

4.1.3 User-specified Control Parameters

For the subgradient procedure, our implementation allows the user to specify the following control parameters (Held, Wolfe and Crowder (1974)): (i) the initial step size multiplier value, (ii) the number of consecutive 'no-improvement' iterations after which the multiplier should be halved, (iii) the maximum number of subgradient iterations, and (iv) the tolerance level to terminate the subgradient procedure when the percentage gap between the upper and lower bounds becomes very small. After some initial testing, we decided to use the following standard parameter settings for all our computational tests:

Initial step size multiplier value = 2

Number of consecutive iterations
after which multiplier is halved = 15

Maximum number of subgradient iterations = 250

Tolerance level = 10^{-9}

4.1.4 Applying Drop Heuristic at Intermediate Stages

As we mentioned previously, applying the drop heuristic to the Lagrangian-based starting solution at the end of each subgradient iteration is very time-consuming. We, therefore, apply the drop procedure only if the current Lagrangian-based starting solution has a lower cost than the previous best starting solution. (Initially, the best starting solution cost is set equal to the cost of the feasible design constructed by the Build method.) However, for some problems where the Build method generates good starting solutions, the drop heuristic may never be applied at intermediate stages because of this rule. Since Lagrangian-based designs are likely to generate good local optima, we force the algorithm to apply the drop heuristic at least once every 100 iterations even if the intermediate starting solutions have higher cost. Thus, if the drop procedure was never initiated in the first 100 iterations, it is applied to the Lagrangian-based starting design constructed from the subproblem solution at iteration number 100. Similarly, the drop procedure is always applied at least twice by the end of iteration 200, and so on.

We implemented a random problem generator to construct test problems for the Lagrangian relaxation algorithm. The problem generator can create test problems with different sizes, arc and commodity densities, cost structures and hop restrictions. To generate a test problem, the user must first specify the following parameters: (i) seed for random number generator, (ii) network size information: number of nodes (n), edges (m), and commodities ($|PQ|$), (iii) cost structure information: weight (w) for the random component in edge costs, and fixed-to-variable cost ratio (r), (iv) average demand information ($\bar{h}$), and (v) hop-restriction information (h_0 for uniform restriction method, and h_1, h_2 for commodity-dependent restriction method).

The program first locates the required number of nodes randomly on a 1000 x 1000 grid and generates a random spanning tree over these nodes (in order to ensure problem feasibility). It then adds the required number of additional edges (i.e., m-(n-1) edges) randomly to the tree. Having constructed the underlying network G:(N,E), the problem generator calculates the fixed and variable cost for each edge in G. Let e_{ij} denote the Euclidean distance between nodes i and j. The fixed cost F_{ij} of edge (i,j) is then computed using the following formula:

$$F_{ij} = (1 - w)\, e_{ij} + w\, \zeta, \qquad (4.1)$$

where w is the user-specified weight ($0 \leq w \leq 1$) and ζ is a random number derived from a uniform distribution with range [0,1000]. Observe that, by varying w, the user can generate cost values that are either completely random (when w = 1) or directly proportional to Euclidean distance (when w = 0); intermediate values of w give a mixture of random and Euclidean costs. We refer to w as the *cost randomness factor*. Note that when w > 0, the fixed costs may not satisfy the triangle inequality.

The variable cost c_{ij} for edge (i,j) is obtained by dividing the fixed cost F_{ij} by the user-specified fixed-to-variable cost ratio r. When r is very small, the fixed costs are negligible compared to the variable costs; in this case, the union of the hop-constrained shortest paths (using the variable costs as arc lengths) for all commodities gives the best design. At the other extreme, when r is very large, the variable costs are negligible, and the best solution is a network with the smallest total fixed cost that contains at

least one path satisfying the hop restriction for every origin-destination pair; in particular, when $h_{pq} = (n-1)$, the optimal solution for large values of r is the minimal spanning tree.

After calculating the edge costs, the problem generator randomly identifies the required number ($|PQ|$) of origin-destination pairs which define the commodities. For each commodity, the demand d_{pq} is derived from a uniform distribution ranging from 0 to $2\bar{d}$ (so that the expected demand for each commodity is the user-specified average demand $\bar{d}$).

Finally, the program generates the hop parameters for each commodity. The user can select either the uniform hop restriction method or the commodity-dependent method. For the first method, every commodity must travel on at most h_0 arcs. The program first verifies problem feasibility by ensuring that G contains at least one path with h_0 or fewer arcs for every origin-destination pair; if not, the user must specify a higher value for h_0. For the commodity-dependent method, the user specifies a window $[h_1,h_2]$ defining the desired range of maximum hop parameters. As discussed earlier, the actual hop parameter h_{pq} for commodity <p,q> is obtained by interpolation (and rounding down) within this range; commodities with higher demands have lower h_{pq} values and vice versa. Again, the program first checks for problem feasibility. If the network does not contain any path from p to q with h_{pq} or fewer arcs, then the value of h_{pq} is incremented by 1 and the feasibility check is reapplied. If the new value of h_{pq} exceeds h_2, the user must increase the upper limit of the window.

Next, we describe the results of our computational experiments to assess the effectiveness of the Lagrangian relaxation algorithm. Subsequently, in Section 4.4, we show how to generate different cost minimizing solutions by varying the hop restrictions.

4.3 Testing the Performance of the Lagrangian Relaxation Algorithm

We use two performance measures to evaluate the effectiveness of the Lagrangian relaxation algorithm: (i) the *% gap* which is defined as the difference between the final (i.e., best) upper and lower bounds expressed as

a percentage of the best lower bound, and (ii) the *computation time* (in CPU seconds) required for the entire procedure. The % gap statistic measures the quality of the lower bounds and heuristic solutions generated by the algorithm since it represents the maximum possible percentage cost differential between the optimal and Lagrangian-based heuristic solutions. Low values of % gaps are desirable since they verify the near-optimality of the heuristic solutions. On the other hand, when the % gaps are high, we cannot accurately assess the quality of the heuristic solutions. The high % gaps may result from two possible causes: either the lower bounds are weak (possibly because of a weak relaxation with an inherently high duality gap) or the heuristic solutions are far from optimal (or both).

Our computational tests attempt to evaluate the effect of three problem characteristics - problem size, cost structure, and hop restrictions - on the algorithmic performance measures (% gap and computation time). *Problem size* depends on the number of nodes in the network, the number (or density) of edges, and the number of commodities. We expect the computation times to increase as problem size increases; however, the % gaps may be lower for problems with dense demand patterns since the total variable cost as a percentage of the total (fixed plus variable) cost is likely to be higher for these problems. The *cost structure* is determined by two factors: the *cost randomness factor* (w), and the *fixed-to-variable cost ratio* (r). We expect problems with higher values of r to be more difficult to solve (since the fixed costs dominate relative to the total variable costs in these problems); the effect of the cost randomness factor is less clear. The third problem characteristic affecting algorithmic effectiveness is the tightness of the *hop restrictions*. We use the average value of h_{pq} over all commodities to measure this tightness. For the uniform hop restriction method, the average hop parameter value is exactly equal to h_0 (the common value of the hop parameter), while $(h_1+h_2)/2$ (the midpoint of the user-specified window) approximates this measure for the commodity-dependent hop restriction method. For convenience, we consider only the uniform hop restriction method in this section; results for the commodity-dependent hop restriction method are reported in Section 4.4. We expect the % gaps to increase as the hop restrictions become tighter. For all test problems, we fixed the average demand value ($\bar{d}$) at 5 units.

4.3.1 Effect of Cost Structure and Hop Restrictions

To isolate the effect of each of the three problem characteristics on algorithmic performance, we first consider different cost structures (cost randomness factors and fixed-to-variable cost ratios) and hop restrictions for a single problem size, namely, a 20-node network with 180 edges and 180 commodities. For this problem size, we generated 3 different network instances (using different random number seeds) with varying cost randomness factors (w = 0, 0.5, and 1); for each network, we applied the Lagrangian relaxation algorithm with different fixed-to-variable cost ratios (r = 10, 20, and 50) and different (uniform) hop parameters (h_0 = 3, 4, and 5 hops). This set of parameter values represents a wide spectrum of possible problem structures, ranging from Euclidean costs to completely random costs, and moderate to high fixed costs (relative to the variable costs).

Insert TABLE I about here

Table I summarizes the results of our analysis for this set of test problems. For each combination of cost randomness factor, fixed-to-variable cost ratio, and hop parameter, this table shows (i) the total fixed cost as a percentage of the total cost in the final heuristic solution (denoted as *% FC*), (ii) the number of edges included in the topology corresponding to the final heuristic solution (*# arcs*), (iii) the weighted average number of hops over all commodities as defined in equation (2.8) (*ave. hop*), (iv) the gap between the final upper and lower bounds expressed as a percentage of the best lower bound (*% gap*), and (v) the total CPU time (in seconds on an IBM 3083) required for constructing the initial heuristic solution, performing 250 subgradient iterations, and applying the drop procedure (*CPU*). The first three statistics describe the characteristics of the final heuristic solution, while the last two measure the algorithm's effectiveness. In general, problems with higher % FC are more difficult to solve; and, final designs that contain a larger number of edges are more reliable since they are likely to contain multiple paths for several commodities.

As expected, the results in Table I show that the % FC and % gaps increase while the number of edges in the final design decreases as the fixed-

to-variable cost ratio increases from 10 to 50. For ratios of 10 and 20, all gaps are less than or equal to 5%. For one problem instance in this category, the algorithm generated the optimal solution and proved its optimality (by generating lower bounds equal to the upper bounds, resulting in a 0% gap); for 8 other instances, the gap was less than 1%. With a fixed-to-variable cost ratio of 50, the gaps are larger (ranging from 4 to 23%). These latter problems represent extreme values of fixed costs, and might have inherently higher duality gaps. We note that, for consistency, all the % gaps reported in Table I correspond to the difference between the upper and lower bounds at the end of 250 subgradient iterations. However, for some sample test problems with high % gaps, we were able to improve the lower bound (and hence reduce the gaps) by running the subgradient procedure for more iterations.

As the hop restriction becomes tighter (i.e., when the uniform hop parameter decreases from 5 to 3), the problems become more difficult to solve and the % gap increases, as does the number of edges in the final design. However, the results do not exhibit any consistent pattern of variation in the % gap (or in the other statistics) as the cost randomness factor changes from 0.0 (Euclidean cost case) to 1.0 (completely random costs).

A detailed examination of the computer outputs showed that the Build method together with the subsequent drop procedure generates good initial upper bounds with very little computational effort. In general, the starting designs constructed by the Build method are sparse relative to the original network (i.e., the all-inclusive design). Dropping 2 or 3 edges from this design leads to a local optimum. The drop procedure reduces the cost of the starting solution by 4.4% on average. The total (Build + drop) CPU time required for generating the initial upper bound is less than 2 seconds in almost all instances. In contrast, if we start with the all-inclusive design (instead of using the Build method), the computation time increases by two orders of magnitude. For example, for the problem instance with $w = 0.0$, $r = 20$, and $h_0 = 4$, the drop procedure required 493 seconds (versus 1.5 seconds when we start with the solution developed by the Build method); it eliminated 154 out of 180 edges in the all-inclusive design before reaching a local optimum. For this problem instance, the Build method led to a slightly better initial heuristic solution (with 0.2% lower cost) compared to the solution generated from the all-inclusive design. In other cases, we found that the

all-inclusive design gave marginally better bounds, but required 300 to 700 seconds of computer time.

Applying the drop heuristic at intermediate stages of the subgradient procedure reduced the initial upper bound by 4.5% on average. The intermediate Lagrangian-based starting solutions were slightly denser than the Build starting solution, though the drop procedure still required only 2 to 8 seconds per application to reach a local optimum in most cases. On average, the drop procedure was initiated about 8 times during the subgradient procedure.

The subgradient procedure requires around 70% of the total computational time, and improves the initial lower bound by an average of 30.6%. The procedure requires more time for problems with less restrictive hop constraints (i.e., with higher h_0) since, for these problems, the hop-constrained shortest path algorithm requires more iterations to solve the routing subproblems. In particular, for values of h_0 equal to 3, 4, and 5, the subgradient procedure required about 80, 110, and 140 seconds, respectively. These computation times show very little variation as the cost randomness factor and fixed-to-variable cost ratio change. The % improvement in the initial lower bound depends noticeably on the fixed-to-variable cost ratios; the improvement is larger for problem with higher ratios. Also, we observed that the % improvement is significantly higher for problems with a mixture of random and Euclidean costs (i.e., with $w = 0.5$) compared to problems with purely random or purely Euclidean costs. This observation suggests that either our multiplier initialization method does not perform as well for problems with $w = 0.5$ or these problems are inherently harder to solve.

We next outline several ways to possibly improve the performance of the Lagrangian-based algorithm.

Improving the Upper and Lower Bounds

The % gaps generated by the Lagrangian-based algorithm can be reduced by searching for better heuristic solutions and by increasing the lower bounds. As we mentioned earlier, running the subgradient procedure for a larger number

of iterations might possibly increase the lower bound. In addition, we might consider strengthening the problem formulation by adding specially-structured valid inequalities (similar to constraint (2.6) of [HCDP]) that further restrict the feasible region of the relaxed problem without eliminating the optimal solution. To devise such inequalities, we require a characterization of optimal solutions that can be translated into constraints for the problem formulation; we also require that these constraints be such that their addition does not significantly increase the computational effort required to solve the Lagrangian subproblems. Previous experience in other problem contexts has shown that some classes of valid inequalities are very effective in reducing the duality gaps (see, for example, Crowder, Johnson and Padberg (1983)). We might also consider other multiplier adjustment methods instead of (or in addition to) subgradient optimization. For instance, we could incorporate a method to reinitialize Lagrangian multipliers at intermediate stages of the subgradient procedure whenever the current incumbent improves (using a method similar to our multiplier initialization option # 3), or devise a dual ascent algorithm.

The upper bounds may possibly be improved by applying the drop heuristic more frequently at intermediate iterations of the subgradient procedure. Recall that, in order to reduce the computation time, our implementation applies the drop heuristic only when the current Lagrangian-based starting solution has a lower cost than all previous starting solutions. This restriction might result in missing some opportunities to generate better incumbents and improve the upper bounds. As with the previous enhancements, this approach can potentially reduce the % gap, but may entail significant additional computational effort. We view these different improvement strategies as fine tuning options that the user might adopt depending on the specific problem instance to be solved.

4.3.2 Effect of Problem Size and Hop Restrictions

So far, we have fixed the problem size and considered the effect of cost structure and hop restrictions on algorithmic performance. We now study the effect of problem size on the quality of bounds and computation time. For this purpose, we fixed the cost randomness factor at 0.5, and the fixed-to-

variable cost ratio at 20. We considered four basic network sizes containing 10, 20, 25 and 30 nodes, respectively. For each network size, we generated sparse as well as dense networks (with correspondingly sparse and dense demand patterns). Table II reports the performance of the Lagrangian relaxation algorithm for 6 test networks with different sizes, for various (uniform) hop restrictions.

Insert TABLE II about here

Once again, the % gaps decrease as the hop parameter increases. In all but three instances, the gaps are under 10%, and the larger gaps can possibly be reduced by increasing the maximum number of subgradient iterations. The % gaps do not show any consistent variation as the network size and density increase; for the 10 node problem the % gap is higher for the denser network, while the converse is true for the 20 node problem. As we might expect, the computation time increases dramatically as the problem size increases.

In summary, our computational results suggest that the Lagrangian-based algorithm is quite effective. For almost all problems with fixed-to-variable cost ratios of up to 20, the method generates tight lower bounds and good heuristic solutions that are generally guaranteed to be within 10 % or less from the optimal solutions. We emphasize that many of our test problems are very difficult to solve optimally. For instance, the integer programming formulation corresponding to the 30-node, 420-edge test problem contains 176,870 integer variables, 176,400 continuous variables, and 882,840 constraints; this problem size is well beyond the realm of capabilities for current state-of-the-art integer programming packages.

4.4 The Network Cost vs Network Performance Tradeoff

The results of the previous section establish the effectiveness of the Lagrangian-based algorithm for generating good cost-minimizing solutions for various parameter settings. We now focus on *solution effectiveness*, namely, the effect of hop constraints on cost, delay and reliability. We illustrate how to generate solutions that define an 'efficient' frontier by

systematically varying the hop parameters. For this purpose, we focus on a single problem instance with 25 nodes, 300 edges, and 300 commodities. Observe that this network is complete (i.e., it contains one edge connecting each pair of nodes); also, the demand pattern is 'complete' (i.e., the set of commodities PQ contains one commodity for each node pair). The cost randomness factor is fixed at 0.5, the average demand at 5 units, and the fixed-to-variable cost ratio at 20.

Insert TABLE III about here

We tested 22 different hop restrictions for this problem instance. The computational results are shown in Table III. The first 7 solutions correspond to uniform hop restrictions, with parameter h_0 ($= h_1 = h_2$) ranging from 1 to 7; the remaining 15 solutions correspond to commodity-dependent hop restrictions with varying means and widths. For example, the hop restriction [2,2] corresponds to a uniform restriction of 2 hops for all commodities. The restriction of [2,5], on the other hand, corresponds to a commodity-dependent restriction with a range width of 3 and mean of 3.5; in this case, high demand commodities are assigned hop parameters close to 2 and low demand commodities are permitted to travel over a maximum of up to 5 channels.

Since we use randomly generated test problems, the actual values of the total cost for different final solutions is not very meaningful. Instead, we consider a *normalized cost* statistic that expresses the actual cost as a percentage of the lowest total cost over all 22 solutions. Thus, a design with a normalized cost of 140 is 40 % more expensive than the lowest cost design obtained with less stringent hop restrictions. Using this normalized cost criterion facilitates our assessment of the cost-benefit tradeoff as the hop restrictions vary.

To measure the average delay performance of the different solutions, we use the *average hops* statistic defined in equation (2.8); this statistic computes the weighted average number of hops over all the selected commodity routings in the final design, where the weights correspond to the respective commodity demands.

Our measure for reliability is *average connectedness*. For a given commodity <p,q>, we define connectedness as the number of alternate arc-disjoint paths from p to q in the final design; it gives a lower bound on the number of arcs that must fail before all p-to-q communication is blocked. The average connectedness statistic is defined as the weighted average number of alternate paths over all commodities; as before, the demand for each commodity serves as its weight for computing the weighted average measure. Thus, high demand commodities are given greater importance in terms of the number of available alternate paths. Our implementation uses a maximum flow routine to compute the connectedness of each commodity in a given solution. If we assign a capacity of 1 to each edge of the final design, then the maximum flow from p to q gives the number of alternate arc-disjoint paths for commodity <p,q>.

As expected, the results of Table III show that cost and reliability increase while delay decreases as the hop constraints become more restrictive. (Recall that we use the mid-point of the user-specified window to measure the tightness of the hop constraints.) In particular, with a uniform hop parameter h_0 of 1 (solution # 1), the design must contain all edges of the original network, every commodity must have a single-hop route, and each commodity has 24 alternate arc-disjoint paths (including the single-hop route). For our problem instance, this design is also the most expensive because of the significant fixed costs. As we relax the hop constraints, the designs become less expensive while the delay and reliability performance deteriorates. The lowest cost design corresponds to the least restrictive (uniform) hop restriction with $h_0 = 7$.

Observe that, as we increase the hop parameters, the final design remains the same beyond a certain point. For example, in the uniform hop case, the solution is unchanged when h_0 increases from 6 to 7. Similarly, with commodity-dependent hop parameters, the same final design satisfies the hop restrictions [3,6] and [3,7]. This pattern arises because, as the hop parameters increase beyond a threshold, the corresponding hop constraints are no longer binding and the same design remains optimal. Finally, we note that for all but one instance the % gap is less than 6.5%, implying that the costs shown in Table III closely approximate the optimal costs for the different hop restrictions.

Insert FIGURE 3 about here

To better assess the cost versus performance tradeoff, Figure 3 shows a graphical display of the delay and cost for different solutions. (Solution # 1 is not shown in this figure since it has a normalized cost of over 400 while the remaining solutions have normalized costs ranging from 100 to 150.) As this figure illustrates, the method generates several 'dominated' solutions. For instance, solution # 2 (with hop restriction [2,2]) dominates solution # 10 (with hop restriction [1,4]) since the latter design has both higher cost and higher delay (and lower reliability). Similarly, solution # 3 dominates solution # 18. The set of 'undominated' solutions define an 'efficient' frontier, as shown in Figure 3, that characterizes the cost-benefit tradeoff when delay specifications are changed. The piecewise linear curve in Figure 3 shows, for instance, that reducing the average number of hops from 1.74 (solution # 2) to 1.71 (solution # 9) increases the total cost by over 16% (relative to the lowest cost design). A similar graphical representation can be used to examine the tradeoff between cost and reliability.

To summarize, our hop-constrained network design model helps to identify a selected subset of good solutions with varying performance characteristics. Using a pictorial representation of these solutions, the network designer can assess the tradeoffs between conflicting criteria and select an appropriate backbone network topology.

5. Conclusions

In this paper, we have proposed a hop-constrained network design model to support long-term network planning exercises. For this model, we developed an effective solution method that can solve relatively large problems. By systematically varying the hop restrictions, we demonstrated the method's ability to generate several alternative cost-minimizing designs with different delay and reliability characteristics. Using the hop constraints to control delays and improve network reliability significantly reduces the complexity of the design problem formulation and enables us to apply an optimization-based solution approach. In addition, the hop-constrained model does not require any assumptions about traffic distributions, nor does it require detailed delay and connectivity specifications. These advantages make the model a useful decision support tool for backbone network design. We believe that our modeling assumptions are reasonable for design purposes in the communication network context. Indeed, our model or its variants might also apply to other large-scale planning problems that arise in logistics and manufacturing systems design; these additional applications are worth exploring.

Several extensions of the basic model merit further investigation. First, though our solution method applies even when the cost functions are piecewise linear and concave, we have not tested its effectiveness for problems with such economies of scale. These problems are harder to solve and the algorithm may possibly require some enhancements to generate good upper and lower bounds for large networks. Second, our model applies to the design of new networks that have no current transmission capacity. Adapting the model and solution method for network expansion planning, where we seek cost effective enhancements to an existing network in order to cope with increasing traffic, is an important next step. To account for the existing network, we require additional capacity constraints in the problem formulation; these constraints make the problem more difficult to solve and might adversely affect the algorithmic performance. Devising methods to systematically vary the user-specified capacity utilization levels and evaluating the impact of these changes on cost and network performance is a third, and potentially fruitful, area for further investigation. Similar to our experiments with hop restriction methods, this study should help the designer select appropriate utilization parameters for different channels. One promising approach is to

use information from the Lagrangian subproblem solutions to gauge the sensitivity of the final designs to variations in channel utilization levels. Finally, it would be interesting to develop and test alternative multiplier adjustment schemes (such as dual ascent) and heuristic initialization and improvement methods to further improve the quality of the upper and lower bounds.

AHUJA, V. 1979. Routing and Flow Control in Systems Network Architecture. *IBM Syst. J.* 18, 298-314.

AITINKEMER, K., and B. GAVISH. 1988. Heuristics with Constant Error Guarantees for the Design of Tree Networks. *Man. Sci.* 34, 331-341.

BALAKRISHNAN, A. 1987. LP Extreme Points and Cuts for the Fixed Charge Network design Problem. *Math. Prog.* 39, 263-284.

BALAKRISHNAN, A., and S. C. GRAVES. 1988. A Composite Algorithm for a Concave-cost Network Flow Problem. To appear in *Networks*.

BALAKRISHNAN, A., T. L. MAGNANTI and R. T. WONG. 1987. A Dual Ascent Procedure for Large Scale Uncapacitated Network Design. Submitted for publication.

BERTSEKAS, D. P. 1980. A Class of Optimal Routing Algorithms for Communication Networks. *Proc. 1980 Int. Conf. on Circuits and Computers*, Atlanta, Georgia.

BERTSEKAS, D. P. 1984. Second Derivative Algorithms for Minimum Delay Distributed Routing in Networks. *IEEE Trans. Communications* COM-32, 911-919.

BILLHEIMER, J., and P. GRAY. 1973. Network Design with Fixed and Variable Cost Elements. *Trans. Sci.* 7, 49-74.

BONUCCELLI, M. A. 1981. Allocating Additional Link Capacities in Computer Communication Networks. IBM Res. Report RC 8967, Yorktown Heights, New York.

BOORSTYN, R. R., and H. FRANK. 1977. Large-scale Network Topological Optimization. *IEEE Trans. Communications* COM-25, 29-47.

CANTOR, D. G., and M. GERLA. 1974. Optimal Routing in a Packet Switched Computer Network. *IEEE Trans. Computers* C-23, 1062-1069.

CHOU, W., F. FERRANTE and M. BALAGANGADHAR. 1978. Integrated Optimization of Distributed Processing Networks. *Nat. Comp. Conf.*, 795-811.

COURTOIS, P. J., and P. SEMAL. 1981. An Algorithm for the Optimization of Nonbifurcated Flows in Computer Networks. *Perform. Eval.* 1, 139-152.

CROWDER, H., E. L. JOHNSON, and M. W. PADBERG. 1983. Solving Large-scale Zero-one Linear Programming Problems. *Oper. Res.* 31, 803-834.

ESWAREN, K. P., and R. E. TARJAN. 1976. Augmentation Problems. *SIAM J. Comput.* 5, 653-665.

FISHER, M. L. 1981. The Lagrangian Relaxation Method for Solving Integer Programming Problems. *Man. Sci.* 27, 1-18.

FRANK, H., and W. CHOU. 1971. Routing in Computer Networks. *Networks* 1, 99-122.

FRANK, H., and W. CHOU. 1972. Topological Optimization of Computer Networks. *Proc. IEEE* 60, 1385-1397.

FRATTA, L., M. GERLA and L. KLEINROCK. 1973. The Flow Deviation Algorithm: An Approach to Store-and-Forward Computer Communication Network Design. *Networks* 3, 97-133.

GALLAGER, R. G. 1977. A Minimum Delay Routing Algorithm Using Distributed Computation. *IEEE Trans. Communications* COM-25, 73-85.

GAVISH, B. 1982. Topological Design of Centralized Computer Networks - Formulations and Algorithms. *Networks* 12, 355-377.

GAVISH, B., and K. ALTINKEMER. 1986. Parallel Savings Heuristics for the Topological Design of Local Access Tree Networks. *Proc. IEEE-INFOCOM 86*, 130-139.

GAVISH, B., and S.L. HANTLER. 1983. An Algorithm for the Optimal Route Selection in SNA networks. *IEEE Trans. Communications* COM-31, 1154-1'61.

GAVISH, B., and I. NEUMAN. 1986. Capacity and Flow Assignment in Large computer Networks. *Proc IEEE-INFOCOM 86*, 275-284.

GAVISH, B., and I. NEUMAN. 1987. Routing in a Network with Unreliable Components. Working Paper, Graduate School of Business, New York University, New York.

GEOFFRION, A. M. 1974. Lagrangean Relaxation and its Uses in Integer Programming. *Math. Prog. Study* 2, 82-114.

GERLA, M. 1973. Deterministic and Adaptive Routing Policies in Packet Switched Computer Networks. Presented at the ACM-IEEE 3rd Data Communications Symposium, Tampa, Florida.

GERLA, M., H. FRANK, W. CHOU, and J. ECKL. 1974. A Cut Saturation Algorithm for Topological Design of Packet Switched Communication Networks. *Proc. Nat. Telecomm. Conf.* NTC-74, 1074-1085.

GERLA, M., and L. KLEINROCK. 1977. On the Topological Design of Distributed Computer Networks. *IEEE Trans. Communications* COM-25, 48-60.

HELD, M., P. WOLFE and H. P. CROWDER. 1974. Validation of Subgradient Optimization. *Math. Prog.* 6, 62-88.

JOHNSON, D. S., J. K. LENSTRA and A. H. G. RINNOOY KAN. 1978. The Complexity of the Network Design Problem. *Networks* 8, 279-285.

KLEINROCK, L. 1964. *Communication Nets: Stochastic Message Flow and Delay* . McGraw-Hill, New York.

KOBAYASHI, H. 1982. Communication Network Design and Control Algorithms - A Survey. IBM Research Report RC 9233, Yorktown Heights, New York.

LAWLER, E. L. 1976. *Combinatorial Optimization: Networks and Matroids.* Holt, Rinehart and Winston, New York.

MAGNANTI, T. L., and R. T. WONG. 1981. Accelerating Benders Decomposition: Algorithmic Enhancements and Model Selection Criteria. *Oper. Res.* 29, 464-484.

MARUYAMA, K., K. FRATTA and D.T. TANG. 1977. Heuristic Design Algorithm for Computer Communication Networks with Different Classes of Packets. *IBM J. Res. Develop.*, 21, 360-369.

MARUYAMA, K., and D.T. TANG. 1976. Discrete Link Capacity Assignment in Communication Networks. *Proc. Third Int. Comput. Comm. Conf.*, 92-97.

PIRKUL, H., and S. NARASIMHAN. 1987. A New Algorithm for the Design of Backbone Networks. Working Paper, College of Business, The Ohio State University, Columbus, Ohio.

PIRKUL, H., and S. NARASIMHAN. 1988. Primary and Secondary Route Selection in Backbone Computer Networks. Working Paper, College of Business, The Ohio State University, Columbus, Ohio.

RARDIN, R. L., and U. CHOE. 1979. Tighter Relaxations of Fixed Charge Network Flow Problems. Technical Report No. J-79-18, School of Industrial and Systems Engineering, Georgia Institute of Technology, Atlanta, Georgia.

ROSSEEL, M. 1968. Comments on a Paper by Romesh Saigal: A Constrained Shortest Route Problem. *Oper. Res.* 16, 1232-1234.

SAIGAL, R. 1968. A Constrained Shortest Route Problem. *Oper. Res.* 16, 205-209.

SEGALL, A. 1979. Optimal Distributed Routing for Virtual Line-switched Data Networks. *IEEE Trans. Communications* COM-27

TCHA, D., and K. MARUYAMA. 1985. On the Selection of Primary Paths for a Communication Network. *Computer Networks and ISDN Systems* 9, 257-265.

TYMES, L. R. W. 1981. Routing and Flow Control in TYMNET. *IEEE Trans. Communications* COM-29, 392-398.

YUM, T. 1981. The Design and Analysis of a Semidynamic Deterministic Routing Rule. *IEEE Trans. Communications* COM-29, 498-504.

TABLE I: Effect of Cost Randomness Factor and FC/VC Ratio on Algorithmic Performance

Network Size: 20 nodes, 180 edges, 180 commodities

RATIO r	Unif. HOP param. h 0	Cost Randomness Factor w														
		0.0					0.5					1.0				
		% FC	# arcs	ave. hop	% GAP	CPU*	% FC	# arcs	ave. hop	% GAP	CPU*	% FC	# arcs	ave. hop	% GAP	CPU*
10	3	16	34	2.26	3.16	116	17	39	2.10	0.69	129	19	37	2.28	1.25	97
	4	12	28	2.54	1.41	129	15	35	2.30	0.55	255	13	28	2.67	0.03	139
	5	10	24	2.74	0.05	180	15	35	2.64	0.57	298	11	25	2.89	0.04	178
20	3	24	31	2.29	3.50	140	24	33	2.22	3.55	147	33	37	2.27	4.28	94
	4	19	25	2.69	4.80	154	24	32	2.40	2.89	170	21	25	2.78	0.65	127
	5	16	22	2.93	0.00	126	20	28	2.59	2.84	320	19	23	2.97	0.95	161
50	3	50	33	2.28	20.18	129	45	32	2.28	19.22	166	56	36	2.29	23.20	100
	4	38	24	2.72	9.19	144	37	26	2.65	11.79	224	43	25	2.80	9.96	135
	5	34	21	3.12	4.13	168	33	23	3.00	8.93	250	36	21	3.23	13.39	162

* CPU time in seconds on an IBM 3083

TABLE II: Effect of Problem Size on Algorithmic Performance

Parameter Settings: Cost Randomness Factor w = 0.50, FC/VC Ratio r = 20

Network size		Uniform HOP parameter h_0														
		3					4					5				
Nodes	Edges **	% FC	# arcs	ave. hop	% GAP	CPU*	% FC	# arcs	ave. hop	% GAP	CPU*	% FC	# arcs	ave. hop	% GAP	CPU*
10	20	47	11	2.15	5.89	3	41	9	2.49	1.94	4	41	9	2.49	4.88	5
10	40	37	12	2.03	9.90	6	35	12	2.14	8.00	9	29	10	2.47	10.70	13
20	90	37	29	2.23	11.47	53	31	25	2.52	5.83	84	31	25	2.50	5.69	101
20	180	24	33	2.22	3.55	147	24	32	2.40	2.89	170	20	28	2.59	2.84	320
25	150	36	44	2.28	11.39	120	29	36	2.55	5.07	365	27	34	2.64	5.28	337
25	300	25	51	2.26	4.23	380	21	43	2.55	1.89	784	19	41	2.68	1.92	1040
30	210	33	58	2.37	8.31	214	26	45	2.78	4.19	386	23	42	2.91	2.70	622
30	420	23	65	2.23	4.53	1916	19	57	2.48	1.65	1399	18	55	2.54	1.43	2195

* CPU time in seconds on an IBM 3083

** Number of Commodities = Number of Edges

TABLE III: Effect of Hop Restriction on Algorithmic Performance

Network Size: 25 nodes, 300 edges, 300 commodities
Parameter Settings: Cost Randomness Factor w = 0.5, FC/VC Ratio r = 20

Soln. No.	HOP parameter $[h_1, h_2]$	Design Characteristics			DELAY	RELIABILITY			% GAP
		Norm. Cost *	% FC	Number of edges	Average # Hops	Connectedness			
						Ave.	Max	Min	
1	[1,1]	412.1	79	300	1.00	24.00	24	24	0.00
2	[2,2]	128.7	39	78	1.74	5.30	9	4	15.51
3	[3,3]	104.9	25	51	2.26	3.50	6	2	4.23
4	[4,4]	100.6	21	43	2.55	2.68	5	1	1.89
5	[5,5]	100.2	19	41	2.68	2.55	5	1	1.92
6	[6,6]	100.0	19	41	2.68	2.55	5	1	1.53
7	[7,7]	100.0	19	41	2.68	2.55	5	1	1.53
8	[1,2]	150.1	45	89	1.62	6.24	10	4	6.38
9	[1,3]	144.9	44	83	1.71	5.78	9	4	4.53
10	[1,4]	141.0	41	75	1.83	5.22	8	4	3.23
11	[1,5]	140.9	40	74	1.86	5.03	8	4	3.27
12	[1,6]	138.0	40	73	1.94	5.02	9	3	2.67
13	[1,7]	137.9	40	71	1.96	4.93	7	3	1.70
14	[2,3]	110.0	30	54	2.15	3.51	6	2	5.41
15	[2,4]	109.0	25	51	2.20	3.37	6	2	5.52
16	[2,5]	109.6	28	55	2.14	3.67	7	2	6.02
17	[2,6]	109.5	27	54	2.18	3.57	6	2	4.92
18	[2,7]	108.8	26	51	2.28	3.40	6	2	6.31
19	[3,4]	101.5	21	44	2.52	2.80	5	1	2.20
20	[3,5]	101.4	22	44	2.51	2.75	5	1	2.08
21	[3,6]	101.3	21	43	2.56	2.68	5	1	3.17
22	[3,7]	101.3	21	43	2.56	2.68	5	1	3.58

* Normalized Cost = Total Cost for design as a percentage of lowest design cost

FIGURE 1: Cost Structure for Channel Capacity

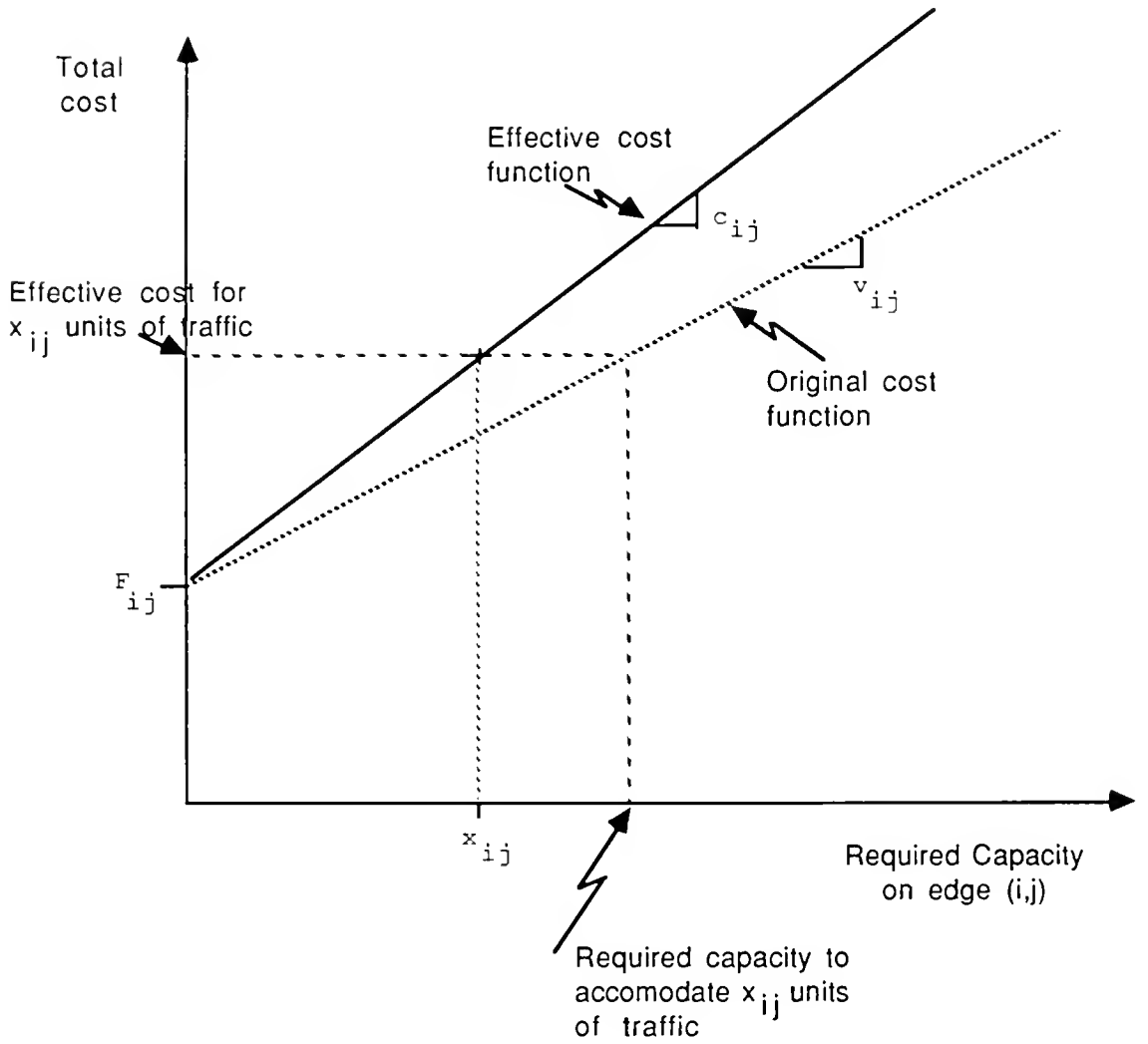

FIGURE 2: Piecewise Linear, Concave Cost Function

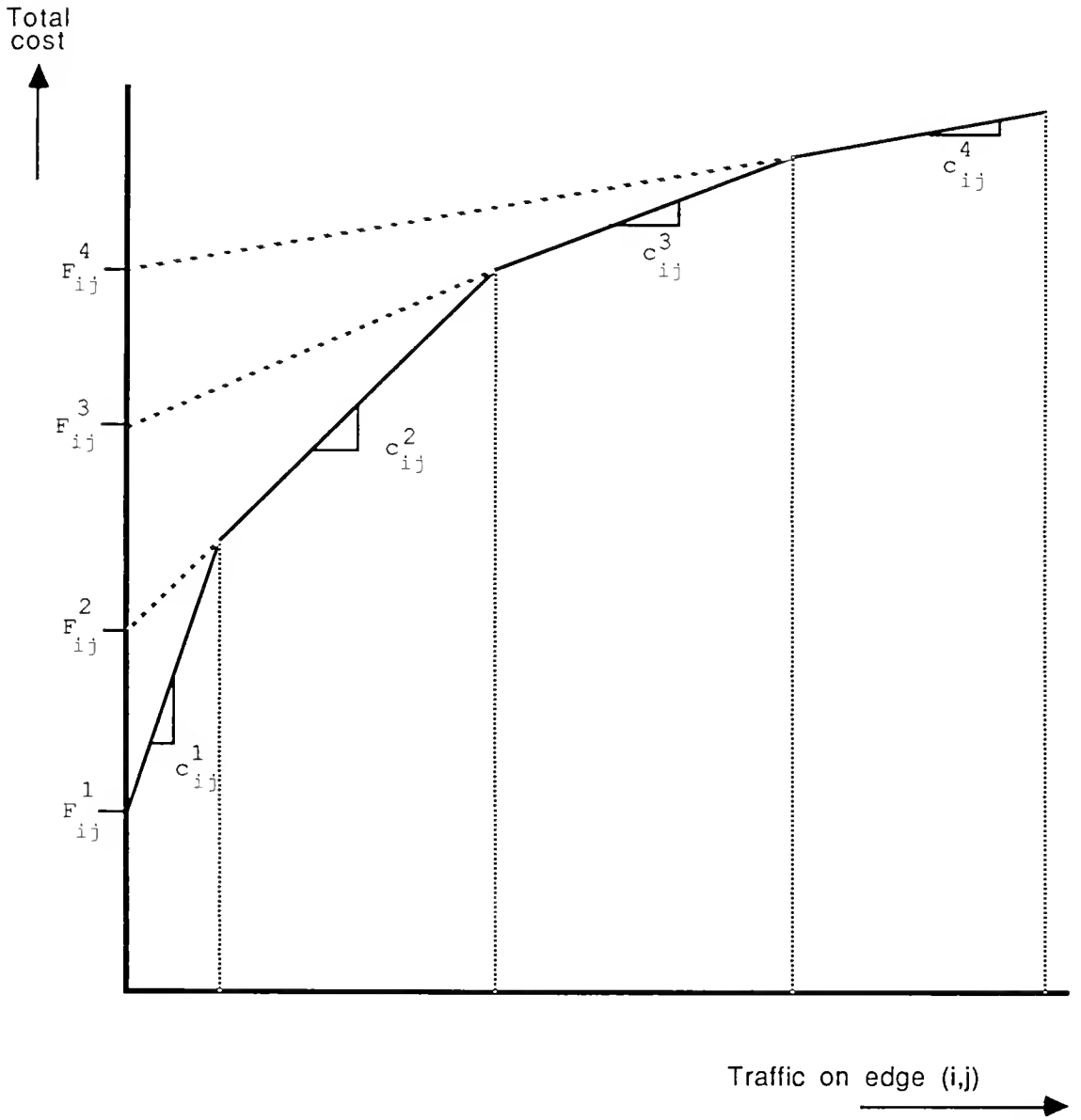

FIGURE 3: COST versus DELAY tradeoff